Plans for Integrated Simulation and Reconstruction Framework

Yury Kolomensky
UC Berkeley

Existing Software Infrastructure

- Much progress made on MECO simulations thus far
  - Two simulations packages, based on Geant3 and Geant4
  - Beamline simulations, backgrounds, detector resolutions and efficiency
  - Reconstruction code
    - L-Tracker Pattern recognition, fitting integrated into GMC
    - Standalone T-Tracker PatRec and Fitter
    - Calorimeter response
- Covered in the previous talks
- A handful of developers, somewhat disjoint package structure

The Next Steps

- Key questions for the experiment
  - ❑ Magnet design and impact on physics capabilities
    - ☞ E.g. field uniformity
  - ❑ Longitudinal vs Transverse Tracker design
  - ❑ Trigger and DAQ development
    - ☞ Calorimeter reconstruction, tracking
- These will require increasingly more sophisticated software capabilities
  - ❑ Detailed simulations
  - ❑ Integrated reconstruction algorithms
  - ❑ Standard benchmarks
    - ☞ Signal and backgrounds
    - ☞ Geometry
    - ☞ Fields

Integrated Simulation/Analysis

MECO

- As the sophistication of the software increase, and more people get involved in the project, overall design issues become important
  - Integration of simulation and reconstruction/analysis
  - Flexibility (various detector packages, physics signatures, backgrounds)
  - Code maintenance and portability
  - Documentation
- Ultimately, would like a system that can be migrated to online and offline operation w/o major redesign

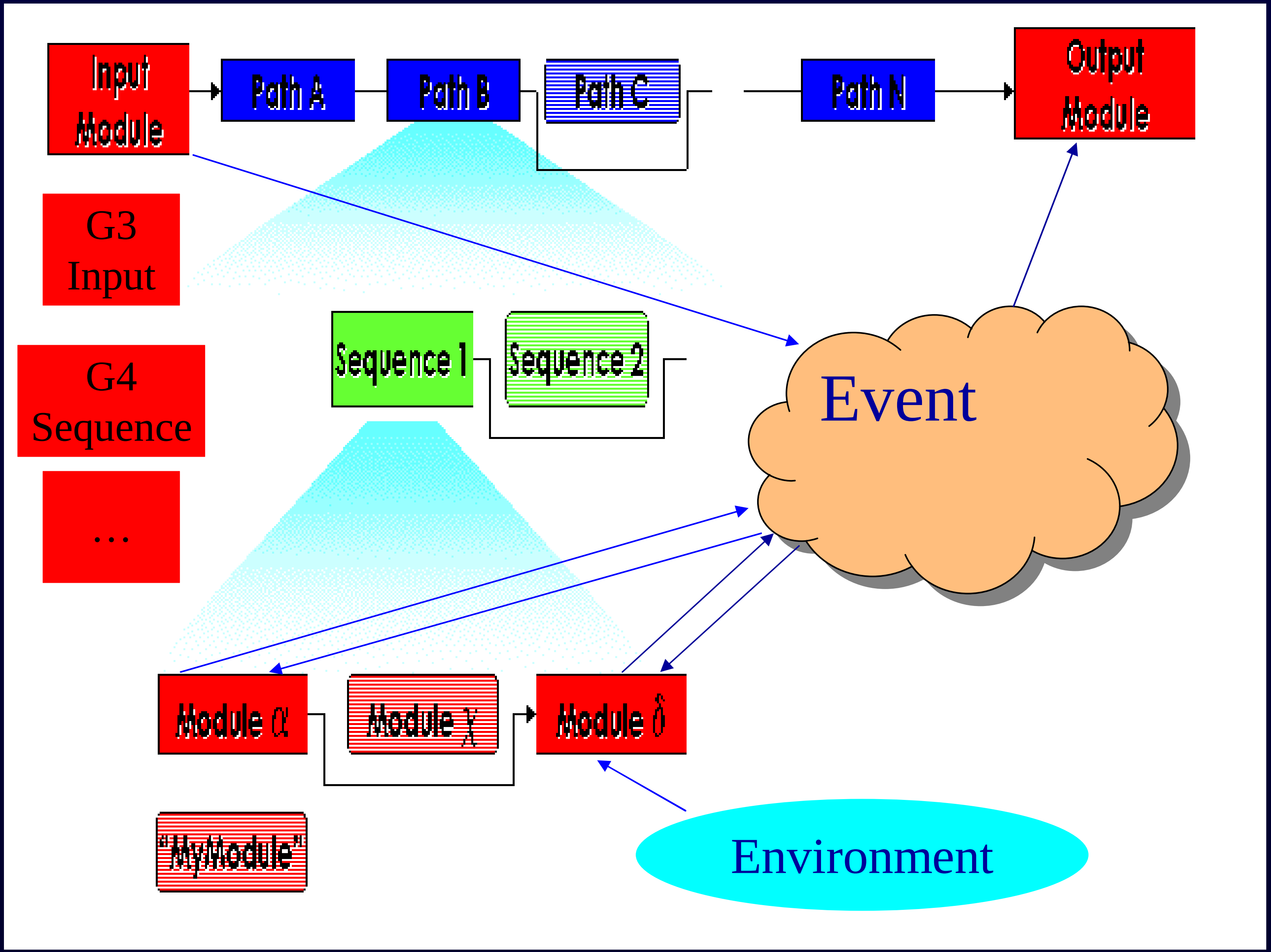
The Framework

- A fairly flexible online/offline system was designed by CDF/BaBar
  - ❑ Based on C++ and tcl scripting language
  - ❑ Modular structure to accommodate different inputs, outputs, execution sequence
  - ❑ Extensive and extendable set of build tools for various architectures (Linux & Solaris, historically supported OSF and HP-UX)
- In various flavors exists in major HEP experiments
  - ❑ BaBar, CDF
- I have a lot of experience with the internals of this Framework, having ported it at least twice (E158, ILC nanoBPM project)
  - ☞ No wheels invented here

The Framework: Best Features

- Execution is organized in *modules*: chunks of code which perform specific tasks
  - ❑ Abstract interface for each module: beginJob(), endJob(), beginRun(), endRun(), event()
  - ❑ Each module is independent of each other (code dependence management: code in parallel) but modules pass data to each other
  - ❑ Execution sequence (which modules are run and in what order) can be changed at run time with tcl scripts
    - ☞ Online, simulation, offline is handled that way
  - ❑ Event structure is extensible
    - ☞ Type-safe interfaces to add/get data to/from event; only modules that directly use particular data objects need to know about them
  - ❑ Extensible Run-dependent environment
    - ☞ Handle "constants" that change slowly in a type-safe manner
  - ❑ Tcl run-time interface
    - ☞ Change parameters of modules, add/remove modules, change inputs/outputs, etc.

The Framework



Code Management

- Code organized in packages
  - Each package is responsible for specific task (e.g. calorimeter digitization, PatRec, etc.)
  - Corresponds to a linkable library
  - Assigned to a responsible person
- By default, code base is in CVS
  - Accessed either by AFS or ssh
  - Revision system: allow parallel development, version control
  - Handles merges, creation/deletion of new files and packages, fallback mechanism

Code Management (cont)

- Regular software releases
  - ❑ Snapshots of software, taken periodically
    - ☞ Every few months
    - ☞ Copies of releases can be either installed locally, or accessed (AFS) from the central location
    - ☞ Each user downloads a small snapshot of the release, checking out only packages they need to recompile
- Build tools: SoftRelTools (SRT) from BaBar
  - ❑ Several OS/compiler architectures
    - ☞ Solaris, RedHat/SL Linux fully functional
  - ❑ Language support for Fortran, C, C++, Java, ROOT shared libraries
  - ❑ Again, no need to reinvent the wheel

Where Do We Start ?

- I'm starting on porting the MECO simulation into The Framework
  - ❑ Inputs: GMC and G4 simulations up to hit creation
    - ☞ GMC inputs through disk files, G4 through either disk files or Framework input modules
    - ☞ Backgrounds through disk files
  - ❑ Digitization: from hits to digital signatures (digis)
  - ❑ Reconstruction
    - ☞ Fortran-based PatRec and C++-based TTracker PatRec
  - ❑ Outputs: ROOT/HBOOK
  - ❑ Other simulation and reconstruction packages to be incorporated (e.g. calorimeter, trigger)

Manpower

- Most of the hard work is actually in subsystems
- Software infrastructure: ~1 FTE
  - Code port/maintenance, optimization, release building, QC/QA
    - ~1 FTE at the start of the project, tail off when operational
  - The rest in subsystem code
    - Estimate ~1 FTE for simulations/reconstruction for each major subsystem (L-tracker, T-tracker, Calorimeter and Trigger, CR shield, magnet), mostly committed
    - Plenty of opportunity for new blood